Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

COTS in the Real World: A Case Study in Risk Discovery and Repair

Scott Hissam
Daniel Plakosh

June 1999

COTS-Based Systems Initiative

Technical Note
CMU/SEI-99-TN-003

This document is available through Asset Source for Software Engineering Technology (ASSET): 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 or toll-free in the U.S. 1-800-547-8306 / FAX: (304) 284-9001 World Wide Web: http://www.asset.com / e-mail: sei@asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218 / Phone: (703) 767-8274 or toll-free in the U.S.: 1-800 225-3842.

Contents

List of Figures

List of Tables

Abstract

Like many organizations in both the public and private sectors, the U.S. Department of Defense (DoD) is committed to a policy of using commercial off-the-shelf (COTS) components in new systems, particularly information systems. However, the DoD also has a long-standing set of security needs for its systems, and the pressure to adopt COTS components can come into conflict with those security constraints. The major elements of this conflict are the DoD's overall approach to system security on one hand and the economic forces that drive the component industry on the other. As DoD managers and system integrators look to the COTS marketplace for components to satisfy more security requirements, this conflict becomes more prominent. In this report, we describe an actual product evaluation where just such a conflict occurred, examine why that conflict exists, and outline the corrective steps that were taken.

Executive Summary

In November 1998 the Joint Engineering Data Management Information and Control System (JEDMICS) program asked the Software Engineering Institute to investigate the use of a particular product for protecting data assets in the JEDMICS system. This product, produced by Cryptek, consists of both networking hardware and software. The objective of the investigation was to identify technical risks in the second phase of JEDMICS deployment of Cryptek products. The first phase of deployment would use Cryptek products for data encryption and firewall protection. The second phase would introduce the use of data labels to provide confidentiality in a multi-user, multi-contractor environment. The first phase of deployment would be "non-intrusive"—i.e., no design or code changes would be needed to the JEDMICS system. The second phase would require design and code changes, but the extent of these changes was unknown. The SEI task was to identify design risks for the second phase deployment, and propose mitigation strategies.

The major findings of this investigation are the following:

1. Platform services needed to use Cyrptek for data labeling work on some JEDMICS platforms (Solaris and IRIX), but not on others (WindowsNT, Windows9x). This means that Windows platforms cannot be used to provide JEDMICS services that submit data to JEDMICS servers where these submissions must use data labels for security.
2. The network protocol used by the JEDMICS system (TCP/IP) does not support changing data labels in the same session. This means that changes to the JEDMICS design and implementation will be required to support clients requesting documents with different data labels.

This report reaches no categorical conclusions regarding the feasibility of using Cryptek products in JEDMICS. We do not know whether labeled data must be submitted from Windows platforms (finding #1). We also do not know how much design or implementation rework is required, or the degree of freedom allowed in making various design tradeoffs—for example, designs that address finding #2 but seriously degrade performance. Also, there are other issues regarding Cryptek that we have not fully investigated—for example, the fact that third-party software such as Oracle, Browsers, etc., need to be made "Cryptek aware" if these products are to use data labels. Thus, while we are not categorical about the feasibility of using Cryptek, we can say there are significant unknowns, at least to the SEI.

The attached report is a technical summary of the SEI investigation. After submitting a draft version of this report to the JEDMICS program office for review, the SEI and JEDMICS outlined a design mitigation strategy (see the Epilogue of this report) and a series of steps that might demonstrate the feasibility of this strategy. However, at this time, the technical feasibility of this mitigation strategy remains unconfirmed.

1 Introduction

"The supreme misfortune is when theory outstrips performance."
—Leonardo da Vinci

Use of commercial off-the-shelf components is becoming more predominant everyday. COTS products are making their way into systems that are being deployed in the U.S. Department of Defense (DoD), federal and state agencies, and U.S. industry. These systems range from the very simple to the very complex, from information systems to embedded systems, and from the non-essential to the most critical. With such widespread use of COTS components, it is easy to see why conflicts between the needs of a system and the capabilities of components can arise. In some cases these conflicts can be so great that there is no resolution, and the COTS solution is abandoned.

This conflict is greatest when products and the standards they implement are outstripped by actual system requirements. A project in the DoD faced such a conflict. This project required the use of encryption and security labels on the data and network traffic that were transmitted from the system. The encryption would ensure that the data could not be seen by unauthorized recipients. The security labels would indicate the level of military-criticality and the proprietary nature of the data and information. A conflict arose between the commercial hardware and software that was used for encryption and labeling.

In this report, we describe the investigations that were performed to determine how well the selected commercial components met the mission needs of the DoD project. We also discusses the underlying principles that were violated, which led to the conflict. The rest of this report is organized as follows: In Section 2, we explain the context of the mission requirements for data labeling and commercial standards employed. In Section 3, we describe the experimental testbed that was created to investigate the commercial component and the conflicts that were discovered. We present our summary in Section 4. In the epilogue (Section 5), we briefly discusses the repair strategy that was enacted to address the findings in this report.

2 Background

The Joint Engineering Data Management Information and Control System (JEDMICS) is a program under joint sponsorship by the armed services of the U.S. Fundamentally, a JEDMICS system is a document and drawing repository. Essentially, one or more document(s)/drawing(s) (referred to generically as "documents" in the remainder of this report) can be stored, requested, and served from a JEDMICS system. Although a JEDMICS system is substantially more complicated than this, such a description will suffice as a working description in this report.

The documents stored on a JEDMICS server often have proprietary commercial data rights or official military classifications and caveats. The data are proprietary because the documents detail the specifications of parts, components, subsystems, and systems in use in the DoD (e.g., nuts, bolts, engines, aircraft). The military classifications and caveats stem from the mission criticality of those items and how they are used by the DoD. The JEDMICS Program Office is not tasked with assigning those data rights and classifications to the documents. However, the program office is required to allow the end user to assign the appropriate labels. Therefore, a JEDMICS system should be functionally capable of labeling documents according to the handling requirements of the installation's cognizant authority.

In support of this requirement, the JEDMICS Program Office investigated the use of commercial-off-the-shelf networking hardware and software to support data labeling. The product selected to support data labeling was Cryptek's Diamond*TEK Ultra*.

2.1 Diamond*TEK Ultra*

Diamond*TEK Ultra* is a network subsystem that is made up of a network interface card (NIC), a user-assigned smart card, and a central management workstation (*DT*Central). The NIC replaces the typical network interface card found in most desktop workstations and servers (e.g., 3Com, NE2000). The product comes with software drivers for the Microsoft Operating Systems, Sun Microsystem's Solaris, and Silicon Graphic's IRIX.

The Cryptek NIC is different from traditional NICs in that it can support a range of security features. The list of security features includes encryption (DES and Type-I crypto) and security data labeling. Additionally, the Cryptek NIC can be programmed to behave in a manner like that of a network firewall (e.g., host and port associated connectivity). The experiments described in this report centered on the data labeling capabilities of the Cryptek NIC.

Basic Operating Concept

Upon computer start, the NIC is configured to operate with a security label. Any network traffic that originates from the NIC is labeled according to the configuration set in the NIC. Conversely, any network traffic received by the NIC is accepted only if the label matches the configuration set in the NIC. Any attempt to generate or receive network traffic that does not match the NIC's configuration is audited to *DT*Central.

NIC Configuration

Configuration of the NIC is performed at any point after the host computer is turned on. This occurs once the end-user inserts his or her assigned smart-card into the NIC's smart-card reader and enters a personal identification number (PIN). The user then selects his or her profile (e.g., secret, unclassified, etc.) via a toggle switch on the card reader. The NIC then sends station identification, user data, and profile information to *DT*Central. *DT*Central performs the necessary table lookups and responds to the NIC with the appropriate configuration. Finally, the configuration is used to program the NIC to the security level(s) for which it is permitted to operate (see Figure 1).

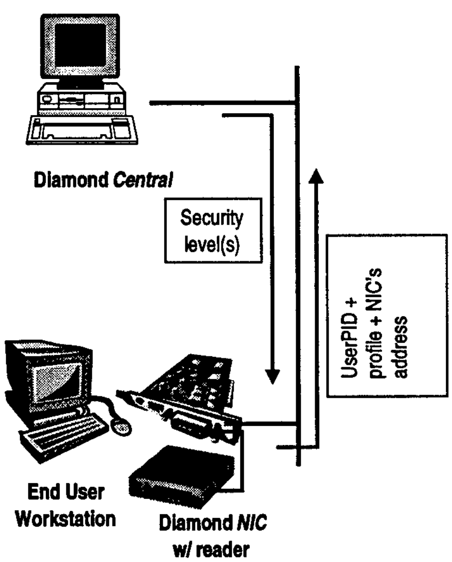


Figure 1. Basic Operating Concept

NIC Operation

Once configured by *DT*Central, the NIC can send and receive network traffic. On a write, the NIC places the assigned security level on every network packet transmitted. This security level is placed in the Internet Protcol's (IP) options field defined in RFC791 (the original protocol specification for IP) as a subtype for security, otherwise know as the Internet Protocol Security Option (IPSO) defined in RFC1108. This security option is simply appended to the end of the normal IP header (see Figure 2).

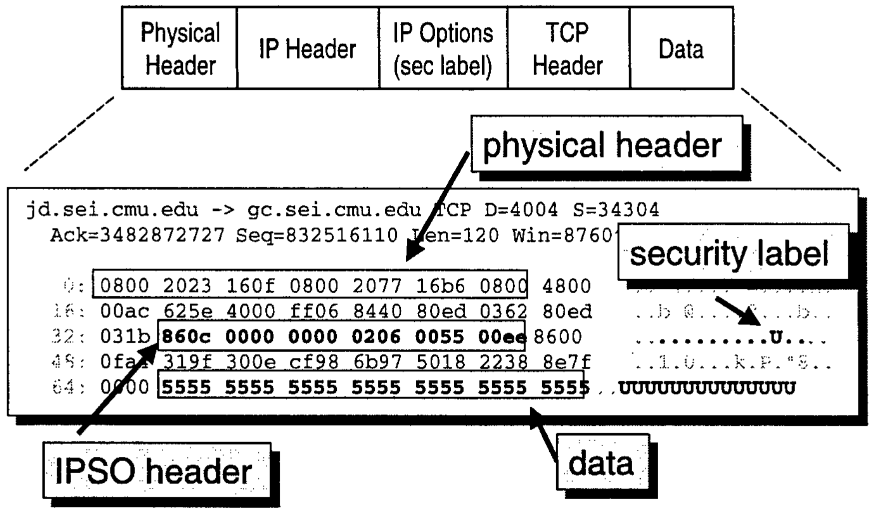


Figure 2. IP Packet with IPSO Option

On a read, the NIC simply queries the IP options field, looking for both the set security option and the contents of that security label. If the label is not appropriate for the configuration set for the receiving NIC, the NIC ignores the IP packet; otherwise the packet is processed accordingly.

Since the option field used for the security data label is actually part of the IP header, the NIC is capable of labeling both TCP (Transmission Control Protocol, RFC793) and UDP (User Datagram Protocol, RFC768) network traffic, as well as any other IP-based protocol.

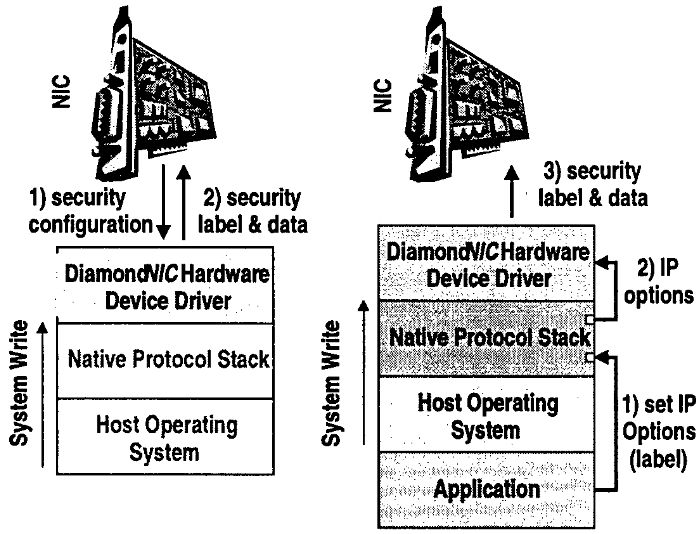
NIC Modes of Operation

The NIC and software drivers can operate in one of two modes: autonomous and non-autonomous. These modes are significant to network write operations, not read operations.

In *autonomous* mode, the NIC labels outgoing IP network traffic with the security level assigned and configured by *DT*Central (discussed above, in "NIC Configuration"). The NIC software device driver reads the assigned security level from the memory on the NIC. This information is then formatted by the software device driver to confirm to the IPSO specification to build a properly-formed IP header with options. On a write, the IP header and options, along with the data, are passed to the NIC for transmission (see Figure 3a). Essentially, in this mode of operation, only a single label is capable of being generated and subsequently transmitted by the NIC.

In *non-autonomous* mode, the data label that is to be placed on outgoing data comes from the user application and not the NIC memory. This label can be set by a user application through a POSIX `setsockopt()` operation (see Figure 3b). Since the NIC is not solely responsible for determining the labeling of the packets in non-autonomous mode, the NIC can transmit

different network packets with different labels—although those labels must still conform to the range configured for the NIC.



a) autonomous mode b) non-autonomous mode

Figure 3. NIC Modes of Operation

2.2 JEDMICS and Diamond*TEK Ultra*

JEDMICS deployment of the Cryptek technology described in Section 2.1 was scheduled to occur in two phases. The Phase 1 deployment is characterized as the *non-intrusive* use of Cryptek in a JEDMICS environment. In this non-intrusive phase, JEDMICS clients and servers would be unaware of Diamond*TEK Ultra*. This means, on one hand, that no software components in JEDMICS needed to be modified. On the other hand, the software would not have the ability to effectively manipulate data labels. Thus, in the Phase 1 deployment, there was no data labeling requirement. The only features of the Diamond*TEK Ultra* product that were used were the encryption and firewall filtering capabilities (essentially creating a hardware-hardened virtual private network). Phase 1 deployment was consistent with the autonomous mode of operation of the Diamond*TEK Ultra* product.

The Phase 2 deployment would add the additional requirement for data labeling, which would require a more intrusive integration of Diamond*TEK Ultra* and the JEDMICS environment. Since JEDMICS servers are required to manage documents with different data labels, a JEDMICS server would assign the correct IPSO label for all documents leaving a JEDMICS server. As such, the server will be expected to reconfigure the IPSO label upon each separate request for a document. Therefore, the non-autonomous mode of operation *seemed* consistent with the concept of JEDMICS operation discussed in Section 2, and *seemed* to match Phase 2 requirements.

There were several open questions about whether Diamond*TEK Ultra* would work in Phase 2, given the requirement for JEDMICS servers to label documents. This suggested the following three lines of inquiry: First, can the IPSO label be manipulated (i.e., set and read)

on server host operating systems where JEDMICS servers run? Second, is the IPSO label set making it into the actual network traffic? Third, are the protocols that are used by the JEDMICS server in any way affected by setting the label? The next section details our investigation.

3 The Investigation

To answer the questions posed in the previous section, we devised three experiments. The first experiment determined the support for the POSIX `setsockopt()` across the operating systems needed in the JEDMICS environment. The second experiment addressed how network packets were affected by insertion of IPSO labels. Finally, the third experiment considered the effect of labeled packets on a network where different hosts had access to different labels.

3.1 Experiment 1—setsockopt()

This experiment centered on the needed support for `setsockopt()` on the operating systems that the JEDMICS server would operate. Without support for `setsockopt()`, it would be next to impossible for a JEDMICS server to directly set IPSO labels programmatically. In this experiment, we looked at Microsoft WindowsNT, Sun Solaris, and Silicon Graphics IRIX.

WindowsNT and IPSO Data Labeling

Microsoft's operating systems have struggled with the Internet Protocol for years, including TCP and UDP; this is still true today. The current version for Microsoft's TCP/IP stack (a.k.a. WinSock) is WinSock version 2.0, which is supported under Windows 95, WindowsNT, and Windows 98.[1] Under WinSock 2.0, there is only limited support for configuration, or parameterization, of the IP, TCP, or UDP protocols. WinSock 2.0 has *no support* for IP_OPTIONS, which is a feature needed to support IPSO data labeling.

A test program and `snoop` under Solaris demonstrated the lack of support for setting various IP parameters, including IP_OPTIONS (Internet Protocol Options) and IP_TTL (Internet Protocol Time To Live). These findings were further substantiated in references [Quinn 98] and [Microsoft 99], where it was confirmed that IP_OPTIONS was not required and may not always be supported. Interestingly enough, attempts to set these options using a test program on WinSock 2.0 *failed silently*. In other words, the WIN32[2] calls to `setsockopt()` and `WSAGetLastError()` returned codes that indicated that the calls were successful, when indeed they were not (see Figure 4). This was confirmed by the `snoop` utility as shown in Figure 5.

1 Previous versions of Windows (3.1, 3.11, or WFW) were not checked.

2 WIN32 is the name of the API which refers to the collection of functions, procedures, system calls which are the underpinnings of recent versions of Microsoft Windows (e.g., Windows95, WindowsNT).

```
1:Winnt_clt -n gc.sei.cmu.edu -e 4004
2:CipsoLabel size is '12'
3:Client connecting to: gc.sei.cmu.edu
4:performing getsockopt(): can we get IP_TTL?
5:  getsockopt() returned: retval 0, ttl 0x20, error 0
6:performing setsockopt() on IP_TTL
7:  setsockopt() returned: retval 0, error 0
8:performing getsockopt(): can we get IP_TTL?
9:  getsockopt() returned: retval 0, ttl 0x40, error 0
10:performing getsockopt(): can we get IP_OPTIONS?
11:  getsockopt() returned: retval 0, lenop 0x0, error 0
12:performing setsockopt()
13:  setsockopt() returned: retval 0, error 0
14:performing getsockopt(): can we get IP_OPTIONS?
15:  getsockopt() returned: retval 0, lenop 0xc, error 0
16:Sent Data [This is a small test message [number 0]]
```

Silent Failure

Figure 4. Sample Execution of WinNT_clt

Referencing line 5 in Figure 4, the test program reports that the current value for the IP parameter time-to-live (TTL) is 0x20 (decimal 32). For line numbers 6 through 9, the test program instructs WinSock to set TTL to 0x40 (decimal 64). It also retrieves TTL from WinSock to confirm that TTL was set accordingly. All operations are performed without error. Lines 10 through 15 use the same WIN32 calls to set the IP_OPTIONS field to the IPSO header, and again all operations are performed without error.

```
pcbj.sei.cmu.edu -> gc.sei.cmu.edu TCP D=4004 S=2701
  Ack=4100106949 Seq=254254011 Len=128 Win=8760

  0: 0800        0060 9761      00 4500    .. #...`.a....E.
 16: 00a8      00 8006 9ab6         80ed    ..U)@.......{..
 32: 031b 0a8d 0fa4 0f27 9bbb f462 aac5 5018    .......'...b..P.
 48: 2238 f552 0000 5468 6973 2069 7320 6120    "8.R..This is a
 64: 736d 616c 6c20 7465 7374 206d 6573 7361    small test messa
 80: 6765 205b 6e75 6d62 6572 2030 5d00 c00c    ge [number 0]...
                   :
176: fa77 1400 0000                             úw....
```

TTL

IHL

Figure 5. A Captured WinSock 2.0 Network Packet

Analysis of the network traffic generated by the test program proved that the calls to update the TTL and the IP_OPTIONS to include the IPSO header did indeed fail. These calls failed silently as the return codes from `getsockopt()`, `setsockopt()`, and `WSAGetLastError()` all returned 0 (zero; or `noError`). Referring to Figure 5, there are three failures that must be noted:

1. Actual TTL of 0x80 (decimal 128) was not correctly reported as shown on line 5 or 9; and was not set to 0x40 (decimal 64) as shown in Figure 4.
2. Although lines 13 and 15 report success in setting and getting the IP_OPTIONS field for the IPSO header, the actual network packet that was sent did not include the IP_OPTIONS set by the test program in line 12 of Figure 4.

3. Actual Internet Header Length (IHL) of 0x05 (decimal 5) confirms that the length of the Internet header did not increase as would be expected if the IP_OPTIONS field had been set. The length of the IPSO header is 12 bytes, or better, three 4-byte words. An IHL of 5 words plus a IPSO header of 3 words should have resulted in a total IHL of 0x08—which was expected, but not found in packet header.

Solaris and IRIX and IPSO Data Labeling

The other operating systems we examined in this experiment behaved correctly and similarly to each other with respect to setting the IP_OPTIONS via `setsockopt()`. However, retrieval via `getsockopt()` of the socket options set behaved differently in each operating system. This was a troubling surprise.

```
/*
 * receiver.c
 */
optlen = sizeof(IpsoLabel);
printf ("size = %d\n", optlen);
status = getsockopt (con_fd, IPPROTO_IP, IP_OPTIONS,
                     (char *) &IpsoLabel, &optlen);
if (status == -1) {
   printf ("getsockopt failed %d, %d\n", status, errno);
   perror ("con_fd");
} else {
   printf ("optlen returned %d\n", optlen);
           printf ("taglevel 0x%02x, category 0x%02x\n",
           IpsoLabel.TagLevel, IpsoLabel.TagCategory);

}
output:
size = 12
optlen returned 16
taglevel 0xff, category 0xee
```

12 correct answer

0x55 correct answer

0xee correct answer

Figure 6. `getsockopt()` Under Solaris

Both Solaris and IRIX would return an additional 4 bytes not associated with the IPSO label. The additional 4 bytes of data returned from the `getsockopt()` function call when requesting the current IP_OPTIONS appears to be a bug that stems from the special processing that is performed when setting the IP_OPTIONS using the `setsockopt()` function call. When setting IP_OPTIONS, special processing is performed to handle source routing. An additional 4 bytes is added to the beginning of the IP_OPTIONS data to account for the first hop when using source routing. This processing appears to occur even when the IP_OPTIONS data do not contain source routing information. The first 4 bytes are not actually included in the IP_OPTIONS data that go out on the wire. However, this bug results in an additional 4 bytes being returned for all `getsockopt()` functions calls requesting the current IP_OPTIONS.

Under Sun Solaris, `getsockopt()` would return the 4 bytes and the data contained in the `IpsoLabel struct` (seen in Figure 6). If the buffer to `getsockopt()` did not account

for this, Solaris would inadvertently overwrite memory not allocated to the `IpsoLabel struct`.

Under SGI's IRIX, `getsockopt()` would also return the 4 bytes not associated with the `IpsoLabel struct` but would not overwrite memory as on Solaris. However, the data returned in the `IpsoLabel struct` was not correct and was offset by 4 bytes.

Results from Experiment 1

1. It is important to note that `getsockopt()` is designed to report on the current IP_OPTIONS settings for outgoing packets and cannot be used to determine the IP_OPTIONS on incoming packets. This test with `getsockopt()` confirmed that JEDMICS would not be able to rely on `getsockopt()` to report on the security label just received. Given the POSIX specification and the confirmed behavior of `setsockopt()` and `getsockopt()` for Solaris and IRIX, it would be impossible for a JEDMICS client to read the actual security label applied to network packets for a document retrieved from a JEDMICS server.
2. `setsockopt()` under Solaris and IRIX did perform as expected as IPSO data labels could be programmatically set.
3. WinSock version 2.0 could not be used at all to support IPSO data labeling because `setsockopt()` had no effect on outbound data packets. Therefore, use of a Microsoft WindowsNT platform as a host for a JEDMICS server will not be possible without significant modification to the WinSock networking stack. Alternatively, it might be possible to find a third-party replacement for native WinSock, but such an investigation was beyond the scope of this effort.

3.2 Experiment 2—IPSO Labels

This experiment was designed to answer the question posed in Section 2: whether or not the IPSO date labels set from experiment 1 were actually making it out into the network traffic. In this experiment, we looked only at an application running under Sun Solaris and Silicon Graphics IRIX.

The Programs

A test harness was constructed to help us answer our questions. This harness has the following components (illustrated in Figure 7 below):

- a simple client/server test program (e.g., sender and receiver) that would run on Sun Solaris and Silicon Graphics IRIX
- an observation program (we'll call it the *sentinel*) to watch network traffic between the sender and receiver looking for network traffic that is labeled, what the label is, and if the label matches the user data contained in the network packet

The sender program simulates the generation of data in a format likely to be generated by a JEDMICS server, mainly the IPSO data label. The receiver is a peer to the sender, which would receive the data and perform some additional tests. The sentinel program acts as an

aid to watch the network traffic between the sender and receiver programs. The details for these programs are discussed below.

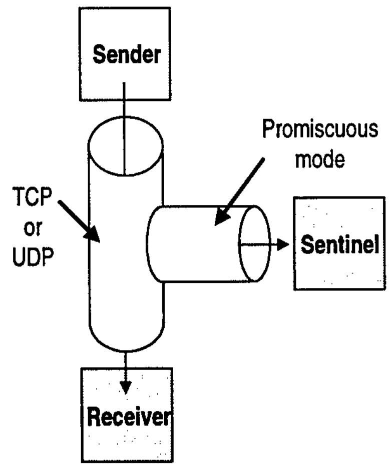


Figure 7. Sentinel Program

Sender and Receiver

The sender program would initiate a connection to the receiver. Once connected, the sender would perform the following functions:

1. Set current security classification to 'U' ('U' for unclassified, 'S' for secret, and 'T' for top secret); the sequence for classifications would follow this ordering.
2. Construct a security label with the current security classification.
3. Use `setsockopt()` to assign the label to outgoing network traffic; check for error.
4. Construct a message that matched the current security classification (e.g., 'U' data for 'U' label, 'S' data for 'S' label).
5. Send data to receiver; check for error.
6. Sequence to the next security classification.
7. Repeat these steps again from step 2 above.

The receiver, as a peer in this connection, would perform the following functions:

1. Block, waiting for data to appear on the inbound socket.
2. Read the data from the inbound socket; report the classification of the data (note: 'U' data is sent with a 'U' label, etc.).
3. Examine the receive buffer to detemine if data of different classifications were intermixed; report any such error.
4. Repeat these steps again from step 1 above.

The sender and the receiver would continue in their respective loops until they were terminated.

The sender took a number of command line options, which could be used to change the characteristics of the experiment. Those options are shown in the following table:

Option	Meaning	Function
`-i`	TCP NO DELAY	Data would be sent immediately and would not permit the operating system to buffer any data before sending.
`-n`	NON-BLOCKING WRITES	Operating system is instructed to copy user data into the kernel and not wait for resources—not all data may be sent due to available kernel resources.
`-o`	RUN ONCE	Program is only to send one message and stop.
`-p [port]`	PORT NUMBER	Sets the port number in which to contact the receiver.
`-s [size]`	BUFFER SIZE	Sets the size of the message to be sent to receiver (1 byte or greater).
`-u`	UDP DATAGRAM	Use UDP datagrams (TCP is the default)

Table 1. Sender Command Line Options

Sentinel

The sentinel program was designed to watch network traffic going between the sender and receiver programs. This was accomplished by programming the network interface to read the network in *promiscuous* mode. Promiscuous mode is simply a mode in which the network interface card (NIC) receives all packets on the network, regardless of the machine to which the packets are addressed. To minimize processing, the sentinel was programmed to read only packets destined for a specific port number for a specific machine.

The sentinel was looking for IP packets that contained *security violations*. A security violation was defined as an IP packet containing an IPSO label that did not match the data contained in the packet. This definition, though limited, was sufficient to cover the cases of security violations for which we were interested. The violations of interest were the following:

- *Packet mislabeling*: TCP or UDP data is mislabeled in the IP packet header. That is, data appears in the network packet with an incorrect data label ('T' data labeled as 'U').
- *Data mixing*: TCP or UDP data requiring two different labels appear in the same IP packet (both 'T' and 'U' data appear in the same packet; regardless of the label).

Packet Mislabeling

Our test harness was worthwhile. It demonstrated a packet mislabeling problem that can be expected in a multi-labeling application (such as a JEDMICS server) running in a UNIX operating system. In UNIX there are two paths to the network interface device, the `write()` system call and the `setsockopt()` library call (which uses the `ioctl()` system call), as shown in Figure 8.

In the first path (labeled *data stream* in Figure 8), blocks of data are moved from user (application) space to kernel space. Data are queued in kernel buffers to be processed by the network stack, then are presented to the network driver, and finally sent though the NIC onto the network. The second path is a direct command path to any functional layer through the kernel, to the network driver or to the NIC (see Figure 8). Such I/O control commands are not queued though the kernel stack like the data stream.

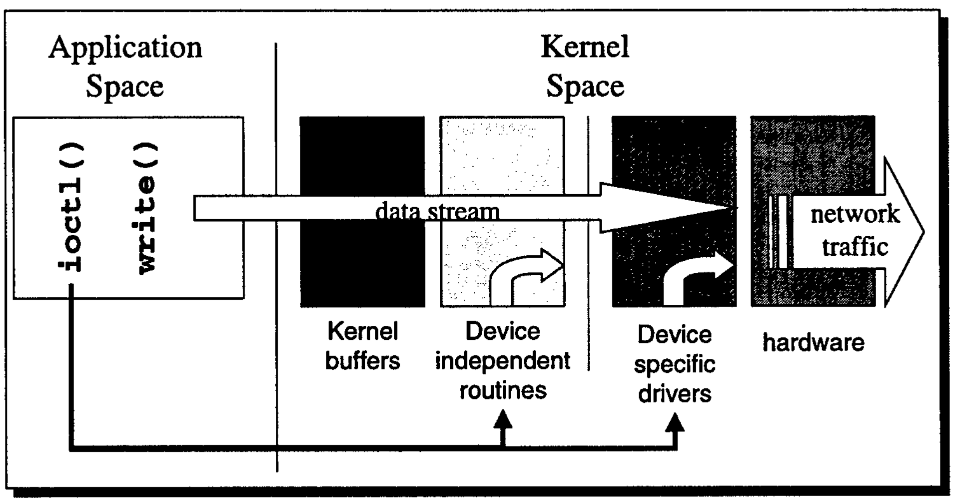


Figure 8. UNIX-Style Network Device Driver Architecture

Because of these two paths to the NIC, an inherent race condition exists that is difficult to predict or control, and is at best problematic. This race condition can be illustrated by considering the following example. While a data stream is making its way through the kernel buffers and protocol stack(s), an I/O control is issued to one of the kernel routines or the device driver before, during, or after the data stream reaches that same logical place in the kernel. In our test harness, this was demonstrated by writing a sequence of 'T's to a network socket and later issuing a `setsockopt()` to label subsequent packets with the label 'U'. In many instances, latency in the kernel writes would result in data streams of 'T's to be labeled by the NIC as 'U' data. This represented a mislabeled network packet (i.e., 'T' data labeled

as ‘U’) and therefore a security violation. This was detected using the native Solaris `snoop` utility and sentinel; a snapshot of that detection is shown in Figure 9.

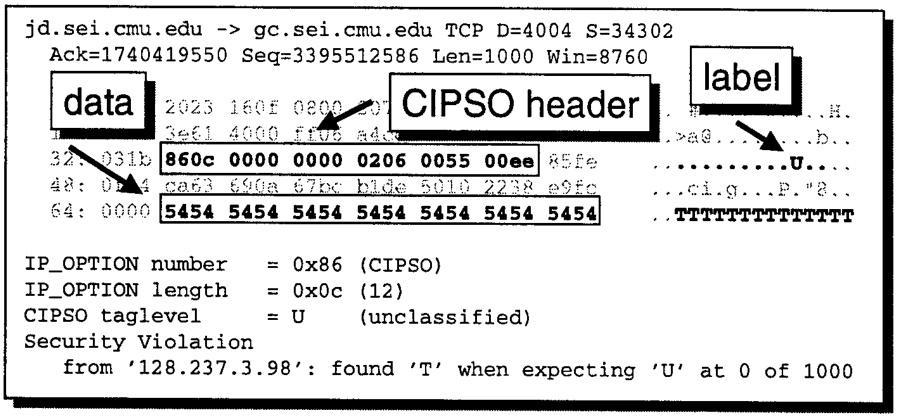


Figure 9. A Captured Mislabeled Network Packet

This race condition existed for the Sun Solaris 2.5.1, Sun Solaris 2.4, and Silicon Graphics IRIX 5.3.

Data Mixing

TCP is a byte-stream protocol in which there are no record markers inserted between application writes. If an application writes 50 bytes followed by a write of 10 bytes, followed by a write of 30 bytes, the receiving application cannot determine the size of each individual write. Additionally, the network system is allowed by the TCP specification to combine data from individual writes into one TCP packet and to fragment a single write into multiple packets. Therefore, there is no way to guarantee how that fragmentation or combination will occur. To illustrate this point, consider the following application-specific protocol over TCP:

1. Client and server connect.
2. Client opens a file.
3. Client writes to the network the total size in bytes of the file to be sent.
4. Client writes to the network the bytes from the file.
5. Client continues from step 2 above until all files have been processed.

In this simple application protocol, it appears to the server that the file data are bounded by the size of the file being sent to it. In fact, the server is easy to write. All it needs to do is read the size of the file from the network, and then proceed to read that number of bytes from the network. Once that number has been reached, the next network read will be the number of bytes for the next file. However, the fragmentation of the original data on the client machine, and the subsequent reassembling on the server machine, can tell a very different story, as shown in Figure 10.

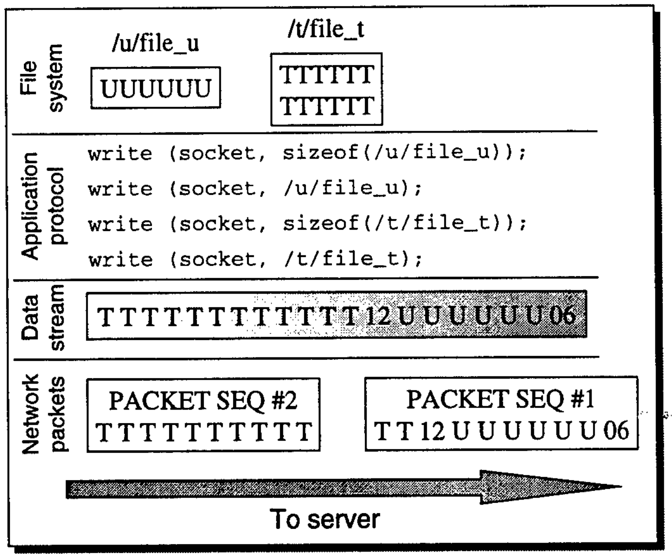


Figure 10. Arbitrary File Boundaries Ignored by TCP

In this example, the data stream in the kernel was arbitrarily broken up into two packets of equal size. By observing the network traffic, we can see that the first packet sequence has data from both file_u (unclassified) and file_t (top secret), where the second packet sequence has the remaining data from file_t. The kernel or the TCP protocol cannot distinguish any artificially constructed boundaries applied to the data. Further, correctly labeling the first packet sequence, with respect to the data contained within it, is also problematic. Neither a 'T' label nor a 'U' label is correct, since both types of data are present.

This problem was also detected using the snoop and sentinel utilities during the lab experiments and is shown in Figure 11.

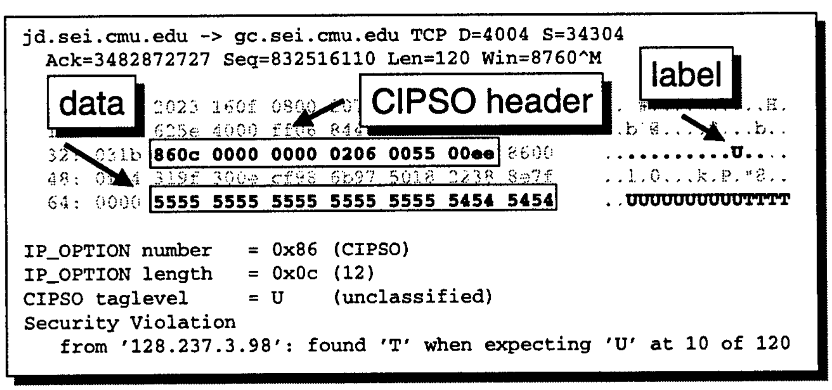


Figure 11. A Captured Data Mixed Network Packet

As illustrated in Figure 11, both 'U' data and 'T' data appeared in a single packet and were detected as a security violation.

Results from Experiment 2

1. TCP/IP as defined by the specification [Postel 81b] does not support multiple data labels over the same TCP connection. Although it is possible to successfully establish and maintain an IPSO data label on a TCP connection, attempts to change the label on that same connection can have non-deterministic results (such as packet mislabeling and data mixing).
2. Interestingly, the sender program that was discussed above (in "Sender and Receiver") was capable of optionally sending UDP datagrams. When the sender and the receiver were configured for UDP we were able to send labeled messages without a security violation—as long as the size of the message did not exceed the maximum transfer unit (MTU) of the NIC. However, if the size of the message exceeded the NIC's MTU, we could observe security violations.

3.3 Experiment 3—Labels and TCP/IP

Finally, we performed an analytical study rather than an experiment conducted in a test harness. Given the evidence gathered from above, it was not difficult to identify cases where IPSO labels in a Cryptek-hardened network could cause problems with TCP connections. Most notably, we cover two such instances here:

- *TCP ACKS and "piggy backing"*: A TCP acknowledgement is combined with TCP data where the data label for the acknowledgement does not match the TCP data. ('U' acknowledgement data label is imposed on 'T' data.)
- *TCP error processing*: TCP packets are dropped by the recipient and are not acknowledged due to a mismatch of the security data labels between sender and receiver.

TCP ACKS and "Piggy-Backing"

The TCP protocol allows acknowledgments (ACKs) to be sent along with data (frequently called piggybacking). Piggybacking usually occurs when one end of a connection needs to simultaneously send data and acknowledge received data. When this occurs, the ACK is sent in the same packet as the data, which results in the same IPSO header for the acknowledgement, and the data. Theoretically, in a secure environment one would expect the IPSO header in an acknowledgment to match the data received.

The following is an example of this risk (as illustrated in Figure 12). Two processes are communicating via TCP. Process 1 sends process 2 'U' data while process 2 sends process 1 'T' data. At the lowest levels of each process' operating system (OS), the TCP drivers much acknowledge (ACK) receipt of data. When process 2's OS prepares the 'T' data for transmission to process 1, the OS is free to piggyback the TCP ACK for 'U' data received from process 1 (at some earlier point in time). In a multi-labeling environment, the label that will be assigned to the "piggybacked" packet cannot be determined. Further, assignment of either a 'U' or 'T' data label would not be correct.

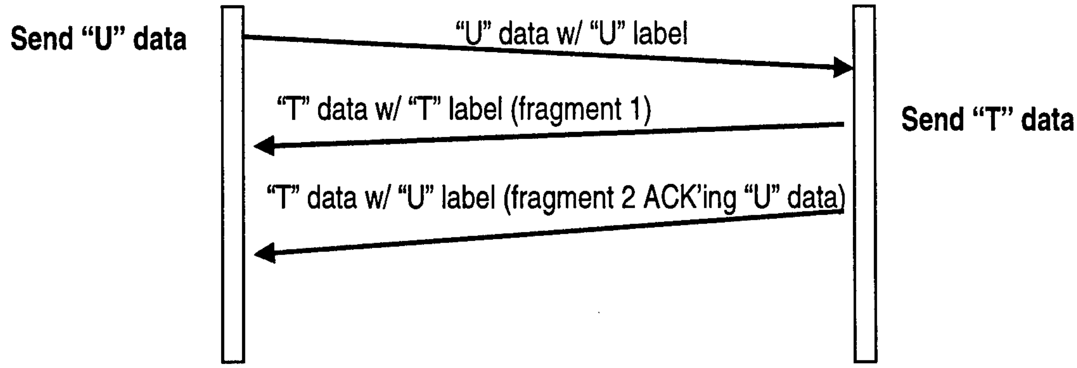


Figure 12. TCP ACKS and "Piggy Backing" Failure

TCP Error Processing

When a TCP packet is lost, the network subsystem does not have to retransmit the identical packet. Instead, the TCP specification allows an implementation to perform re-packetization, which is the ability to send a packet that contains the lost segment and additional data that need to be sent. Due to possible re-packetization of lost segments and the inability to specify the IPSO header for a specific segment of data, one cannot reliably set the IPSO header to reflect the data being sent once transmission of data begins.

When an incoming or outgoing packet violates a security constraint, the NIC would silently discard the packet without notifying the application or operating system. This feature will cause problems for TCP if a packet is discarded by the NIC once a connection has been established (see Figure 13). Because TCP is a reliable protocol, it will resend the dropped packets as the card will continue to drop the same packets that violate the security constraints. Eventually the TCP connection will be broken due to retransmission failures. The application has no way of determining the cause of the failure (e.g., was it a TCP circuit failure or a security violation).

In further analysis, we concluded that this behavior was not only acceptable, but preferred, because it prevented covert channel analysis, which is in itself a potential security violation.

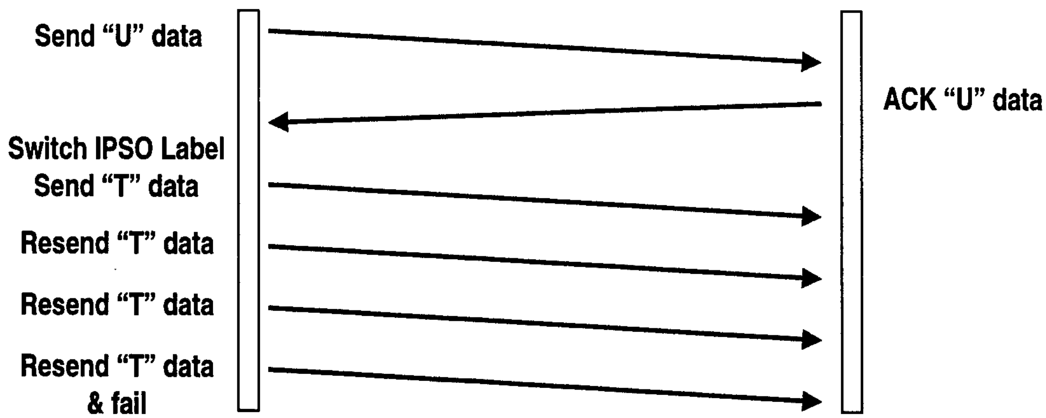


Figure 13. TCP Error Processing

Results from Experiment 3

Although not directly observed in a testbed, analysis based on an understanding of the TCP protocol indicates other problems associated with the application of multiple IPSO data labels on the same TCP connection. The problems associated with TCP protocol acknowledgements and error processing would be difficult to work around without a fundamental change to the TCP protocol (which is unlikely).

4 Summary Findings

Our investigations identified two issues that need to be addressed to successfully integrate Cryptek's Diamond*TEK Ultra* technology for Phase 2 of deployment into the JEDMICS environment. The issues are

- limitation of TCP support for switching IPSO data labeling over a single TCP-based connection (discussed in Sections 3.2 and 3.3)
- limitation of WindowsNT support for IPSO data labeling (discussed in Section 3.1)

We found that the application of this technology in the context of JEDMICS servers that are required to perform multiple data labeling functions was in direct conflict with the underlying protocols and intended use of IPSO data labeling. Without a fundamental change in the behavior of TCP, there is little likelihood that the application of the off-the-shelf Cryptek technology could be applied in the context originally defined by the JEDMICS environment.

Applications of data labeling are subject to the capabilities and limitations assigned to IPSO. Essentially, the IP is capable of carrying security classification information on IP packets (and also TCP segments), so this information can be communicated end-to-end across multiple networks. In this manner, packet-level security information permits hosts and gateways that operate in multilevel secure environments to properly segregate packets for security considerations [Postel 81b].

For TCP, IPSO data labeling is limited to operate on a per connection basis. This means that once a TCP connection is established, the IPSO label initially given to that connection is supported for the life of that connection. Resetting the IPSO label a second time for the same TCP connection is not supported by the specification: Section 2.9 (Precedence and Security) of the TCP specification in [Postel 81b] warns that the use of IPSO to provide precedence and security information is limited to a "per connection basis to TCP users."[3] Further Postel goes on to state that not all TCP modules will necessarily function in a multilevel secure environment.

These statements, combined with the experiments performed by the Software Engineering Institute (SEI), illustrate this limitation of IPSO to be true. Commercial operating systems do not support changing the data label during a connection. The experiments conducted and explanations given in Section 3 illustrate the nature of this limitation.

[3] Users, in this context, are higher level applications and application-specific protocols based on TCP.

5 Epilogue

The JEDMICS program believed that application of IPSO labeling via Cryptek's technology would be an attractive, seamless mechanism for satisfying the requirements placed upon the program. This theory would have been born out if JEDMICS servers had to deal only with single classification levels. However, the additional level of complexity of requiring support for different labels for each document had a negative cascading effect through the application of Diamond*TEK Ultra*. Such application needs were in conflict with the standards set for IP networks nearly 20 years ago and with the implementation of a single vendors network stack, which fell short of the TCP specification.

To resolve this mismatch, the JEDMICS program changed the context of the problem they were trying to solve. This was accomplished by the introduction of a security processor (possibly to be prototyped by Phase 2 deployment). Figure 14 illustrates how this security processor would operate.

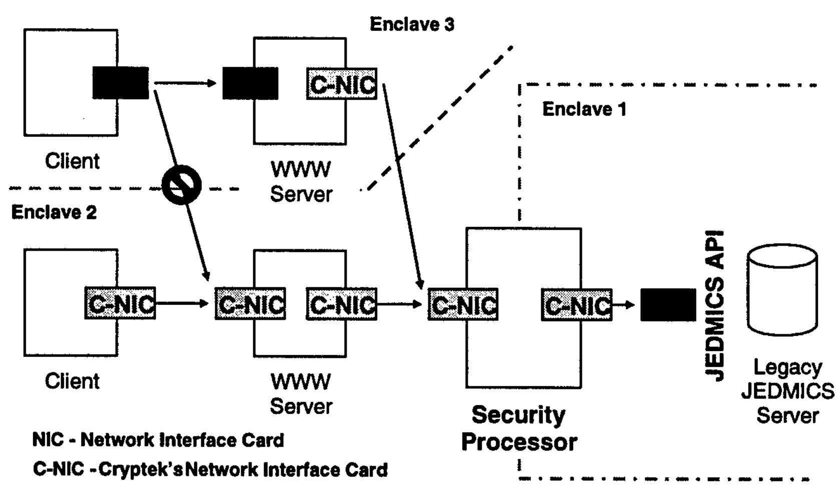


Figure 14. Security Processor Example

In the example shown above, a security processor would be the sole gateway into a JEDMICS server. In this case, the security processor would use a native application-specific API (i.e., JEDMICS API for a JEDMICS server) to access documents. The security processor at the application layer would read meta-data available from the legacy system via the API, noting the security label. In turn the security processor would make use of Diamond*TEK Ultra* to appropriately label outgoing data according to the information contained in the meta-data.

Given that the data outbound from the security process are labeled accordingly, Diamond*TEK Ultra* would perform the necessary functions to encrypt the data and enforce data labeling.

Given this capability, the security processor would be able to service multiple enclaves, which would be required to use Diamond*TEK Ultra*. In the example above, if a WWW server from enclave 3 requests data and information from the legacy system (through the security processor), which is required to be labeled higher than that enclave is permitted access, the security processor can deny that data without any changes being made to the legacy backend server. This enforcement would also be supported by Diamond*TEK Ultra*, as data labeled higher than the receiving system is permitted to receive would be denied and audited by *DT*Central.

References

[DoD 85] DoD 5200.28-STD, "DoD Trusted Computer System Evaluation Criteria" [online]. U.S. Departmentof Defense, December 1985. Available WWW: <URL: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html>.

[IETF 93] IETF CIPSO Working Group. "Common IP Security Option Version 2.3" [online]. Trusted Systems Interoperability Group, March 1993. Available WWW: <URL: http://www.tsig.org/tsig/working_groups/cipso/cipso-rfc.txt>.

[Kent 92] Kent, S. "U.S. Department of Defense Security Options for the Internet Protocol" (RFC-1108) [online]. Information Sciences Institute, November 1992. Available WWW: <URL: http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1108.txt>.

[Microsoft 99] Microsoft Corporation. *Microsoft's Visual Studio Documentation*, references on IP_OPTIONS [online]. Microsoft Corporation, 1999. Available WWW: <URLs: http://premium.microsoft.com/msdn/library/sdkdoc/sockspi/wsanxref_58fm.htm

http://premium.microsoft.com/msdn/library/devprods/vs6/vc++/vcmfc/_mfc_casyncsocket.3a3a.getsockopt.htm

http://premium.microsoft.com/msdn/library/devprods/vs6/vc++/vcmfc/_mfc_casyncsocket.3a3a.setsockopt.htm>.

[Postel 81a] Postel, J. "Internet Protocol, DARPA Internet Program Protocol Specification" (RFC-791) [online]. Information Sciences Institute, September 1981. Available WWW: <URL: http://info.internet.isi.edu:80/in-notes/rfc/files/rfc791.txt>.

[Postel 81b] Postel, J. "Transmission Control Protocol, DARPA Internet Program Protocol Specification" (RFC-791) [online]. Information Sciences Institute, September 1981. Available WWW: <URL: http://info.internet.isi.edu:80/in-notes/rfc/files/rfc793.txt>.

[Quinn 98] Quinn, B. "WinSock Version 2.0: Overview, Status and Pointers" [online]. Bob Quinn, March 1998. Available WWW: <URL: http://www.sockets.com/ws2_stat.htm>.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (LEAVE BLANK)

2. REPORT DATE
June 1999

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
COTS in the Real World: A Case Study in Risk Discovery and Repair

5. FUNDING NUMBERS
C — F19628-95-C-0003

6. AUTHOR(S)
Scott Hissam, Daniel Plakosh

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-99-TN-003

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12.A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Like many organizations in both the public and private sectors, the U.S. Department of Defense (DoD) is committed to a policy of using commercial off-the-shelf (COTS) components in new systems, particularly information systems. However, the DoD also has a long-standing set of security needs for its systems, and the pressure to adopt COTS components can come into conflict with those security constraints. The major elements of this conflict are the DoD's overall approach to system security on one hand and the economic forces that drive the component industry on the other. As DoD managers and system integrators look to the COTS marketplace for components to satisfy more security requirements, this conflict becomes more prominent. In this report, we describe an actual product evaluation where just such a conflict occurred, examine why that conflict exists, and outline the corrective steps that were taken.

14. SUBJECT TERMS
case study, commercial off-the-shelf (COTS), data labels, integration, security

15. NUMBER OF PAGES
24 pp.

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102